Corese : a Corporate Semantic Web Engine

Olivier Corby, Catherine Faron Zucker

Corese : a Corporate Semantic Web Engine

Olivier Corby[1] Catherine Faron-Zucker[2]

(1) INRIA, Sophia Antipolis, France
(2) I3S, University of Nice-Sophia Antipolis, CNRS, France

2002, April 18th

Abstract

With the aim of building a Corporate Semantic Web, the content of the documents must be explicitly represented through metadata in order to enable contents-guided search. The Corese engine is dedicated to the querying of corporate semantic webs whose documents are described into RDF annotations. Corese interprets these RDF metadata in the Conceptual Graphs (CG) model in order to exploit the inference capabilities of this formalism. This paper presents our mapping of RDF into CG and its interest in the context of a Corporate Semantic Web.

Introduction

Corporate Semantic Webs are a promising application of the Semantic Web technology. When limited to a particular domain, a Web can be associated with an ontology describing the concepts of its domain so that the Web documents may be semantically annotated by using this ontology. The main research topic of our team is to study and develop such Corporate Semantic Webs in order to implement Corporate Memories. The Corese engine developped in our team is dedicated to the querying of corporate semantic webs whose documents are described into RDF annotations. Corese interprets these RDF metadata in the Conceptual Graphs (CG) model in order to exploit the inference capabilities of this formalism. Previous works have studied the similarities between the RDF(S) model and the Conceptual Graphs model [Corby et al. 2000][Delteil et al. 2001]. In this paper, we first present our mapping of RDF(S) into CG. We then present the annotation language of Corese: RDF(S) extended with CG features providing a more expressive language for the representation of the ontological knowledge. Then we present the query language of Corese. Finally we present the three main applications in which Corese has been involved and our approach validated.

1 RDF(S) and Conceptual Graph Models

1.1 The Conceptual Graph Model

A conceptual graph [Sowa 94, Sowa, 1999] is a bipartite (not necessarily connected) graph composed of concept nodes, and relation nodes describing relations between these concepts. Each concept node c of a graph G is labeled by a couple $< type(c), referent(c) >$, where referent(c) is either the generic marker * corresponding to the existential quantification or an individual marker corresponding to an identifier; M is the set of all the individual markers. Each relation node r of a graph G is labeled by a relation type type(r); each relation type is associated with a signature expressing constraints on the types of the concepts that may be linked to its arcs in a graph.

Concept types (respectively relation types of same arity) build up a set Tc (resp. Tr) partially ordered by a generalization/specialization relation $\geq$ (resp. $\leq$). (Tc, Tr, M) defines the support upon which conceptual graphs are constructed. A support thus represents a domain ontology.

The semantics of the Conceptual Graph model relies on the translation of a graph G into a first order logic formula thanks to a ϕ operator as defined in [Sowa, 1984]: $\phi(G)$ is the conjunction of unary predicates translating the concept nodes of G and n-ary predicates translating the n-ary relation nodes of G; an existential quantification is introduced for each generic concept.

Conceptual graphs are provided with a generalization/specialization relation $\leq$ corresponding to the logical implication: $G_1 \leq G_2$ iff $\phi(G_1) \Rightarrow \phi(G_2)$. The fundamental operation called projection enables to determine the generalization relation between two graphs: $G_1 \leq G_2$ iff there exists a projection π from G_2 to G_1. π is a graph morphism such that the label of a node n_1 of G_1 is a specialization of the label of a node n_2 of G_2 with $n_1 = \pi(n_2)$. Reasoning with conceptual graphs is based on the projection, which is sound and complete with respect to logical deduction.

1.2 Mapping of the RDF(S) and CG models

The RDFS [RDF 1999, RDFS 2000] and CG models share many common features and a mapping can easily be established between RDFS and a large subset of the CG model. An in-depth comparison of both models is studied in [Corby et al., 2000].

Both models distinguish between ontological knowledge and assertional knowledge. First, the class (resp. property) hierarchy in a RDF Schema corresponds to the concept (resp. relation) type hierarchy in a CG support; this distinction is common to most knowledge representation languages. Second, and more important, RDF properties are declared as first class entities like RDFS classes, in just the same way that relation types are declared independently of concept types. This is this common handling of properties that makes relevant the mapping of RDFS and CG models. In particular, it can be opposed to object-oriented approaches, where properties are defined inside of classes.

In both models, the assertional knowledge is positive, conjunctive and existential; it is represented by directed labeled graphs. An RDF graph G may be translated into a conceptual graph CG as follows:

Each arc labeled with a property p in G is translated into a relation node of type p in CG. Each node labeled with an identified resource in G is translated into an individual concept in CG whose marker is the resource identifier. Its type corresponds to the class the identified resource is linked to by a *rdf:type property* in G.

Each node labeled with an anonymous resource in G is translated into a generic concept in CG. Its type corresponds to the class the anonymous resource is linked to by a *rdf:type* property in G.

Regarding the handling of classes and properties, the RDF(S) and CG models differ on several points. However these differences can be quite easily handled when mapping RDF and CG models.

RDF binary properties versus CG n-ary relation types: the RDF data model intrinsically only supports binary relations, whereas the CG model authorizes n-ary relations. However it is possible to express n-ary relations with binary properties by using an intermediate resource with additional properties of this resource giving the remaining relations [RDF, 1999].

RDF multi-instantiation versus CG mono-instantiation: the RDF data model supports multi-instantiation whereas the CG model does not. However, the declaration of a resource as an instance of several classes in RDF can be translated in the CG model by generating the concept type corresponding to the most general specialization of the concept types translating these classes.

Property and relation type signatures: in the RDF data model, a property may have several domains whereas in the CG model, a relation type is constrained by a single domain. However, the multiple domains of an RDF property may be translated into a single domain of a CG relation type by generating the concept type corresponding to the most general specialization (according to the new RDF semantics) of the concept types translating the domains of the property.

Both models allow a way of reification. Extensions to RDF(S) have been proposed to define contextual knowledge [Delteil et al., 2001a, 2001b].

Managing RDF as Conceptual Graphs can be seen as :

- the compilation of the type hierarchy in an orthogonal dimension (the cg support),
- the association of a compiled type to each resource.

The resulting model provides an optimized processing of queries based on a compiled type hierarchy.

2 Annotation with RDF and CG

The Corese language enables for the representation of both the assertional knowledge embedded in the annotations and the ontological knowledge upon

which the annotations are built is RDF and RDFS, translated into Conceptual Graphs and their support. In this section, we first detail the translation of an RDF annotation into a CG when representing assertional knowledge with Corese. We then describe the representation of ontological knowledge in the Corese language.

2.1 Representation of assertional knowledge

Regarding the differences between RDF and CG models, the translation of an RDF annotation into a CG requires the handling of multi-instanciation and multiple inheritance.

In RDF, a resource can have several types. For example, the resource below has types `Engineer` and `PhDStudent`:

```
<s:Engineer rdf:about='http://www.inria.fr/acacia/Alexandre.D'/>
<s:PhDStudent rdf:about='http://www.inria.fr/acacia/Alexandre.D'/>
```

In the CG model, a concept is an instance of a single type. When translating RDF into CG we compute (generate) on the fly for each RDF resource the greatest common subtype of its types, we add this type to the concept type hierarchy and maintain the consistency, and we declare the concept as an instance of this computed type. In the example above, we would generate the following type:

```
<rdfs:Class rdf:ID='Engineer_PhDStudent'>
    <rdfs:subClassOf rdf:resource='#Engineer'/>
    <rdfs:subClassOf rdf:resource='#PhDStudent'/>
</rdfs:Class>
```

The resource would then internally be typed as :

```
<s:Engineer_PhDStudent
     rdf:about='http://www.inria.fr/acacia/Alexandre.D'/>
```

This handling of multiple types ensures that the projection operation returns relevant results. If we ask for an engineer or a PhDStudent, the projection returns the concept. If we ask for a resource that is engineer and PhDStudent, the projection also returns the concept. When output, such a concept is printed as an instance of the native types, thus hiding the existence of the internal common subtype. The concept above is printed as a resource of type Engineer and PhDStudent, i.e. a resource having two types:

```
<s:Engineer rdf:about='http://www.inria.fr/acacia/Alexandre.D'>
    <rdf:type rdf:resource='&s;PhDStudent'/>
</s:Engineer>
```

2.2 Representation of ontological knowledge

The Corese language for the representation of the ontological knowledge is based on RDFS which is extended to enable the representation of metaproperties and of axiomatic knowledge.

In some applications, a good IR precision depends on the declaration of properties on properties. We have extended the RDF Schema model with three meta-properties called transitive, symmetric and reflexive with boolean values and rdf:Property as domain. These properties are defined in the corese namespace. It is up to the system to compute all the tuples of a relation.
The following example shows the definition of the *relative* property which is declared to be transitive, symmetric and reflexive.

```
<rdf:Property rdf:ID='relative'>
   <rdfs:domain rdf:resource='#Person'/>
   <rdfs:range rdf:resource='#Person'/>
   <cos:transitive>true</cos:transitive>
   <cos:symmetric>true</cos:symmetric>
   <cos:reflexive>true</cos:reflexive>
</rdf:Property>
```

Below is the extension of the RDFS metamodel that enables the definition of such properties :

```
<?xml version="1.0" ?>
<!DOCTYPE rdf:RDF [
<!ENTITY cos   "http://www.inria.fr/acacia/corese#">
<!ENTITY rdf   "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY rdfs  "http://www.w3.org/2000/01/rdf-schema#">
]>

<rdf:RDF xmlns:rdfs="&rdfs;" xmlns:rdf="&rdf;"  xmlns:cos="&cos;">

<rdf:Property rdf:ID='&cos;transitive'>
    <rdfs:domain rdf:resource='&rdf;Property'>
    <rdfs:range  rdf:resource='&rdfs;Literal'/>
</rdf:Property>

</rdfs:RDF>
```

Symmetry and reflexivity are built on the same pattern. Transitive and symmetry closure are computed after loading of annotations and resulting edges are included into the graph.

Reflexivity is computed at query time. For each reflexive relation in the query, Corese computes the reflexive relation tuples candidates in the target graph and considers them during query processing. They are discarded after query processing completes.

Inverse property. It is also interesting to specify the inverse property of a given property and to generate the inverse property value in annotations. We have defined an extension of the RDF Schema to provide this definition, as shown below.

```
<rdf:Property rdf:ID='&cos;inverse'>
    <rdfs:domain rdf:resource='&rdf;Property'>
    <rdfs:range  rdf:resource='&rdf;Property'/>
</rdf:Property>
```

Below is an example of an inverse property :

```
<rdf:Property rdf:ID='hasMember'>
    <rdfs:domain rdf:resource='#Organization'/>
    <rdfs:range  rdf:resource='#Person'/>
    <cos:inverse rdf:resource='#isMemberOf'/>
</rdf:Property>
```

From this definition, the Corese engine infers the definition of the inverse isMemberOf property.

2.3 Inference Rules

An ontology can contain axioms and rules that enable to deduce new knowledge from existing one. But RDF Schema does not provide such a mechanism. Hence we have proposed an RDF Rule extension to RDF & RDFS.

Corese has an inference engine based on forward chaining production rules. The rules apply on Conceptual Graphs and can enrich a graph according to the conclusions. For example, the rule below states that if a person ?p is head of team ?t which has person ?p has a member, then person ?m manages person ?p:

```
IF [Person: ?m]-(head)-[Team: ?t]-(hasMember)-[Person: ?p]
THEN [Person:?m]-(manage)-[Person: ?p]
```

The rules are applied once the annotations are loaded and before query processing occurs. Hence, annotations are augmented by rules.

2.3.1 Conceptual Graph Rules

According to [Salvat and Mugnier, 1996] we consider a rule $G_1 \Rightarrow G_2$ as a pair of lambda abstractions $(\lambda x_1, ..., \lambda x_n G_1, \lambda x_1, ..., \lambda x_n G_2)$ where the x_i are co-reference links between generic concepts of G_1 and corresponding generic concepts of G_2 that play the role of being the rule variables.

A rule $G_1 \Rightarrow G_2$ applies to a graph G if there is a projection π from G_1 to G, i.e. G contains a specialization of G_1. The resulting graph is built by joining G and G_2 while merging each $\pi(x_i)$ in G with the corresponding x_i in G_2. Joining the graphs may lead to specialize the types of some concepts, to create relations between concepts and to create new individual concepts (i.e. *concept without a variable*).

2.3.2 Rule Syntax

The rule syntax is based on the RDF/CG mapping. As a rule is of the form:

```
IF CG1 THEN CG2
```

and as RDF can be interpreted as CG, the syntax of the rules eventually uses RDF:

```
IF RDF1 THEN RDF2
```

where RDF1 (resp. RDF2) is the RDF markup for CG1 (resp. CG2).

Hence, we propose the following syntax for RDF rules, with the convention that variables are prefixed by '?', and are local to the rule:

```
<cos:rule>
<cos:if>
    <s:Person rdf:about='?m'>
        <s:head>
            <s:Team rdf:about='?t'>
                <s:hasMember>
                    <s:Person rdf:about='?p'/>
                </s:hasMember>
            </s:Team>
        </s:head>
    </s:Person>
</cos:if>

<cos:then>
    <s:Person rdf:about='?m'>
        <s:manage rdf:resource='?p'/>
    </s:Person>
</cos:then>
</cos:rule>
```

In case the rule have several conditions (resp. conclusion), several `cos:if` (resp. `cos:then`) markup may occur.

2.3.3 RDF Schema for Corese RDF Rule language

```
<rdfs:Class rdf:ID='&cos;rule'/>

<rdf:Property rdf:ID='&cos;if'>
    <rdfs:domain rdf:resource='&cos;rule'/>
</rdf:Property>

<rdf:Property rdf:ID='&cos;then'>
    <rdfs:domain rdf:resource='&cos;rule'/>
</rdf:Property>
```

3 The Corese Query Language

The Corese query language is RDF with addition of some conventions to introduce variables and operators. An RDF query statement is interpreted as a query conceptual graph and is processed by a CG projection of the query on the annotation graphs.

A variable is prefixed with a question mark. The query below returns the title of Work resources, because the Title property is given the value '?t' which is a variable:

```
<s:Work s:Title='?t'/>
```

It is possible to refer to another resource by means of a variable. For example, return ?c, the creator of a Work :

```
<s:Work s:Creator='?c'/>
<s:Person rdf:about='?c'/>
```

It is possible to test whether the same resource has several types :

```
<s:Person   rdf:about='?p'/>
<s:Employee rdf:about='?p'/>
```

3.1 Comparisons

We can compare values with constants. For example, "find a Work the title of which is 'XML in a nutshell" " is expressed by:

```
<s:Work s:Title='"XML in a nutshell"'/>
```

We can compare a value with a constant using an operator. For example, "find a Work the date of which is greater than 1789" is expressed by:

```
<s:Work s:Date='>1789'/>
```

Possible comparators are :

```
numeric and string (alphabetic order) : <= , <, >=, >

string :
        =
        ~       contain, ignoring case
        ^       starts with
        %       match regular expression
type :
       <:       strict subtype
      <=:       subtype or equal
       =:       equal type
      >=:       supertype or equal
```

```
       >:       strict supertype

negation : ! (negation of an operator, e.g. !~)

or :      |
```

Note that within XML syntax, the < character must be written `<` in the operators.

3.2 Comparison of values of properties

The query "Find two resources with the same date" is expressed by:

```
<s:Work   s:Date='?d'/>
<s:Person s:Date='?d'/>
```

The query "Find a Person with a later date that a Work" is expressed by :

```
<s:Work   s:Date='?d'/>
<s:Person s:Date='>=?d'/>
```

When several operators are present, they are implicitly connected by an AND unless separated by the OR operator noted `|`.

3.3 Comparison of types

The standard projection returns resources that are subtypes of the requested type. For example, if one queries for Vehicle, one gets all subtypes of Vehicle in addition to Vehicle itself. However, it is sometimes interesting to specify more precisely the type of resource one wants to retrieve. Hence, it is possible to compare the type of a resource with a given type. For example, one may be interested in resources of type Vehicle which are not Airplane. This can be written:

```
<s:Vehicle rdf:about='!<=:Airplane'/>
```

The statement `!<=:Airplane` means : "with a type which is not a subtype nor is equal to Airplane".

The statement below searches for exact instances of Airplane:

```
<s:Vehicle rdf:about='=:Airplane'/>
```

3.4 Querying the Ontology

The ontology may be queried using the same query language, i.e. RDFS with variables. In order to be queried, the schema must be loaded as a conceptual graph. For example : find a class the label of which contains 'Sapiens' and return its super classes :

```
<rdfs:Class rdfs:label='~Sapiens' rdfs:subClassOf='?s'/>
```

4 Discussion

The Conceptual Graph framework enables us to implement main RDF(S) features : RDF graph, resources typed by a schema, multiple inheritance of classes, multi instantiation (one resource may has several types), hierarchy of properties, new conjunctive semantics of domain and range, multiple inheritance of properties with inheritance of conjunction of signatures and reification.

Furthermore, the CG projection operator smoothly enables to implement a search engine by matching a query graph with target graphs.

Eventually, CG enable to introduce a natural and simple rule language for RDF, based on CG rules.

Corese serves as semantic search engine in the Comma project, Corporate Memory Management through Agents. Comma is a European IST project involving Deutsche Telekom, ATOS Origin, CSTB, University of Parma, LIRMM and INRIA, [Gandon 2001, D2, D13]. The system was built on top of the Jade FIPA compliant multi agent platform from University of Parma [Jade].

Comma relies on a Corporate Memory RDF(S) Ontology and exploits RDF metadata describing resources, written according to this schema. The ontology contains 470 concepts, 75 relations and a terminology in french and in english.

As another example, Corese can load and process the Gene Ontology [GO] which contains about 10000 concepts and 75000 relations.

5 Conclusions

Corese is an RDF(S) processor based on Conceptual Graphs. It enables to load RDF Schema and RDF data and process information retrieval queries on the data as well as on the schema. Several extensions of RDF(S) enhance the Corese annotation language, enabling to process both transitive, symmetric, reflexive, and inverse properties, and inference rules to enrich the annotation base.

Corese has been used in two real-world applications: the CoMMA project and the Samovar project. Corese has also been used in the Samovar project, a design project memory at the Renault car company. Corese has been involved in the Escrire, an academic project dedicated to the comparison of objects, description logics and conceptual graphs for metada representation.

These three projects have shown the validity of our approach of translating RDF(S) into CG for query-processing in Corporate Semantic Web.

Bibliography

[Berners-Lee, 2000] Reflections on Web Architecture Conceptual Graphs and the Semantic Web `http://www.w3.org/DesignIssues/CG.html`

[Chein and Mugnier, 1997] M. Chein, M.L. Mugnier. Positive Nested Conceptual Graphs. In Proc. of the 5th International Conference on Conceptual Structures (ICCS'97), Seattle, WA, USA, LNAI 1257, Springer Verlag, p. 95-109, 1997.

[Comma] Corporate Memory Management through Agents. European IST poject IST-1999-12217.
http://www.si.fr.atosorigin.com/sophia/comma/Htm/HomePage.htm

[Corby et al., 2000] O. Corby, R. Dieng, C. Hébert. A conceptual graph model for W3C Resource Description Framework. In Proc. of the 8th International Conference on Conceptual Structures (ICCS'00), Darmstadt, Germany, LNCS 1867, Springer-Verlag, 2000.

[Delteil et al., 2001a] A. Delteil, C. Faron and R. Dieng. Learning Ontologies from RDF Annotations. In Proc. of IJCAI'01 Workshop on Ontology Learning, Seattle, USA, August 2001.

[Delteil et al., 2001b] A. Delteil, C. Faron and R. Dieng. Extensions of RDFS Based on the Conceptual Graph Model. In Proc. of the 9th ICCS'2001, Stanford, CA, USA, LNAI 2120, Springer-Verlag, p. 275-289, 2001.

[D2] CoMMA Consortium. Definition of scenarios, requirements and evaluation process. 2000.

[D13] CoMMA Consortium. Final Multi-Agent System. 2001.

[Gandon 2001] F. Gandon, Experience in Ontology Engineering for a Multi-Agents Corporate Memory System, in Proceedings Workshop "Ontologies and Information Sharing" IJCAI 2001, Seventeenth International Joint Conference on Artificial Intelligence August 4th - 10th, 2001, Seattle, Washington, USA, pp. 119-122

[GO] Gene Ontology Consortium http://www.geneontology.org/

[Jade] Java Agent DEvelopment Framework, University of Parma,
http://sharon.cselt.it/projects/jade

[RDF, 1999] Resource Description Framework Model and Syntax Specification, 1999. W3C Recommendation.

[RDFS, 2000] Resource Description Framework Schema Specification, 2000. W3C Candidate Recommendation.

[Salvat and Mugnier, 1996] E. Salvat, M.L. Mugnier. Sound and complete forward and backward chainings of graph rules, In Proc. of the 4th ICCS'96, Sydney, Australia, LNCS 1115, Springer-Verlag, p. 248-262, 1996.

[Sowa, 1984] J.F. Sowa. Conceptual structures: information processing in mind and machine. Addison-Wesley, Reading, Massachusetts, 1984.

[Sowa, 1999] J.F. Sowa. Conceptual Graphs: DpANS.
http://concept.cs.uah.edu/CG/Standard.html 1999.